Surfel Stripping

Tamy Boubekeur, Patrick Reuter, Christophe Schlick

Surfel Stripping

Tamy Boubekeur* Patrick Reuter+ Christophe Schlick*
*: LaBRI - INRIA - CNRS - University of Bordeaux +:LIPSI - ESTIA
[boubek|preuter|schlick]@labri.fr

Figure 1: The Asian Dragon can be rendered at its full definition of 3 609 601 points, with antialiased 2D texturing and cube mapping, at 31 frames per second at a display resolution of 1600x1200 pixels.

Abstract

This paper presents an efficient combination of techniques for fast stripping and multiresolution rendering of Point-Based Surfaces (PBS) called Surfel Stripping. Surfel Strips are small triangle strips that interpolate the PBS. There are two major contributions. First, at loading time, we efficiently convert the PBS into triangle strips. This is done by first generating a set of overlapping small triangular meshes that interpolate the PBS, then removing redundant triangles and finally stripping the small triangular meshes by using a cache-friendly stripping method. All these operations are performed by using an octree data structure. Second, we reuse this data structure for providing a multiresolution interactive visualization of the surfel strips at rendering time. Since Surfel Stripping is local and very fast, it can be used in a lot of situations as an object-space alternative to the image-space surface splatting and thus be considered half way between point-based rendering and local polygonal generation. Rendering Surfel Strips is very efficient since it neither requires multi-pass rendering nor time-consuming vertex/fragment shaders compared to surface splatting. We show also how to exploit the locality of the surfel strips for maintaining compatibility with point-based modeling tools, such as local deformations of surfaces. We finally give some examples of well known visual enrichments developed for polygons, directly applied to PBS thanks to surfel strips.

Keywords: Point-based graphics, fast surface conversion, triangle stripping, multiresolution rendering, graphics data structure

1 Introduction

The interest for *Point-Based Surfaces* (PBS) has grown significantly in recent years in the computer graphics community. Several authors have already explained the reasons of this popularity [Alexa et al. 2004], e.g. the widespread use of 3D acquisition devices that directly generate PBS, or the riddance of connectivity management that greatly simplifies many algorithms and/or data structures. It is now widely admitted that when including additional information at each point (such as normal vectors, colors or material properties) and using specific rendering techniques (mainly to efficiently fill the holes that may appear between the points), PBS can become as flexible as the ubiquitous polygonal surfaces. Following Pfister et al. [Pfister et al. 2000], such enriched points are commonly called *surfels*.

A large variety of rendering techniques for PBS have been presented in the literature. Most of them are based on *splatting*, where a reconstruction kernel (e.g. gaussian convolution) is centered at each projected point to fill the neighboring pixels. The accumulation of the contributions from all the kernels can be considered as an image-space surface reconstruction that is generated on the fly. This approach has a lot of advantages such as noise filtering and antialiasing, and thus enables high-quality rendering. Unfortunately splatting also involves a totally different graphics pipeline, compared to the one used in current 3D graphics hardware. As a consequence, even advanced hardware implementations of splatting techniques [Botsch et al. 2005] have to resort to expensive combinations of vertex shaders, fragment shaders and multi-pass rendering to finally obtain a surface that could have been rendered directly if its equivalent polygonal expression were available.

Starting from this observation, we propose an efficient object-space alternative based on a set of small pieces of triangulated surfaces that we call *Surfel Strips*. Surfel Strips can be quickly generated while loading the PBS either from a local disk or from some network, and they are stored in a specific octree-based data structure, the *Stripping Tree*. Note that, despite the use of triangles for rendering, Surfel Stripping cannot (and should not) be considered as a point-to-mesh reconstruction technique, since we never generate explicit connectivity between neighboring Surfel Strips. In other words, the core representation of objects is still the point cloud. The Surfel strips, are just used for the rasterization, and since they are purely locally generated, they can be locally updated during some point-based modeling session, where common point-based tools [Pauly et al. 2003] are used to modify the shape of the 3D object.

We consider that the technique is located half way between pure mesh reconstruction techniques and pure point-based rendering techniques (a complete discussion on this topic can be found in

Section 5).

The remainder of this paper is organized as follows: Section 2 recalls some previous work with a special focus on rendering and fast conversion of PBS, Section 3 details all the steps involved in the technique we propose, Section 4 presents some experimental results obtained on various point-based models. In Section 5, we discuss the advantages and drawbacks of our approach compared to existing ones, and finally, in Section 6, we briefly describe some work in progress and propose some directions for future research.

2 Previous work

The basic idea to use points as rendering primitives can be attributed to the seminal paper of Levoy and Whitted [Levoy and Whitted 1985]. However, rendering a sufficiently large amount of points at interactive framerates only became feasible when an efficient point-based rendering system was presented by Grossman and Dally [Grossman and Dally 1998]. Their work initiated a highly growing interest towards point-based graphics, and we refer the reader to [Alexa et al. 2004; Kobbelt and Botsch 2004] for a complete survey of the so-called *Point-Based Graphics* techniques. As our paper is somewhere between rendering and fast *lazy* local reconstruction, we propose to focus more precisely on point-based rendering techniques and fast polygonal generation techniques.

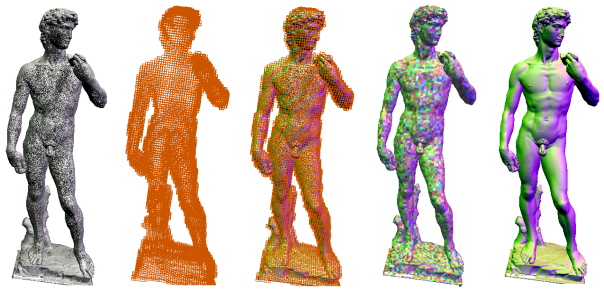
Point-based rendering: Even if some recent techniques try to get the best of both worlds, we think that point-based rendering can still be divided into two main families. First, there are quality-oriented approaches, which are mainly based on powerful filtering techniques. One of the early papers in that family is undoubtedly the work by Pfister et al. [Pfister et al. 2000]. This work has then been extended by Zwicker et al. [Zwicker et al. 2001], with the *EWA Surface Splatting*, one of the most popular point-based rendering techniques, which is based on the screen space formulation of the Elliptical Weighted Average (EWA) filter, initially proposed by Heckbert for antialiased texture mapping on polygonal meshes [Heckbert 1986]. EWA splatting provides high-quality anisotropic filtering, and EWA splats can be efficiently rendered on programmable GPUs [Ren et al. 2002; Botsch and Kobbelt 2003; Guennebaud and Paulin 2003; Botsch et al. 2004; Botsch et al. 2005].

Second, there are performance-oriented approaches, which are mainly based on specific data structures for efficient rendering of very large point sets, such as 3D scanned objects. The early member of this family is the QSplat technique developed by Rusinkiewicz et al. [Rusinkiewicz and Levoy 2000] as part of the Digital Michelangelo Project [Levoy et al. 2000]. That kind of technique has also been used in hybrid point-polygon rendering systems [Dachsbacher et al. 2003; Chen and Nguyen 2001; Cohen et al. 2001; Dey and Hudson 2002; Coconu and Hege 2002; Gobbetti and Marton 2005]. These techniques **do not** propose a solution to the so-called hole filling problem. Actually, their basic principle is rather to use a point-based representation to provide an efficient level-of-detail rendering for complex polygonal meshes, than to provide a true rendering solution for point-based surfaces.

Fast polygonal generation: Combined with their introduction of *Point Set Surfaces* based on the Moving Least Squares (MLS) approximation technique, Alexa et al. [Alexa et al. 2001] implemented a first point-based rendering technique quite related to ours, rendering a PBS as a collection of overlapping two-dimensional parametric patches that locally approximate the surface. For every patch, a quad mesh is generated by sampling the parametric domain of the bivariate polynomial. Since the patches are generated independently, it is obvious that the resulting surface is not C^0 continuous. Moreover, as neighboring patches do not share common normal vectors on boundaries, a visual smoothness for the rendered surface is only achieved when employing a very large number of patches. After that, various approaches were proposed to fit a conservative polygon soup to point clouds. In order to quickly generate polygonal primitives of a point cloud, Linsen et al. have proposed the Fan Clouds [Linsen and Prautzsch 2003], in which for each surfel a triangle fan is constructed on its k-neighborhood. Many fast local triangulation schemes have been proposed for surface reconstruction. In particular, we will reuse the idea of lower dimensional meshing [Gopi et al. 2000; Boubekeur et al. 2005]. Recently, Wicke et al. have proposed a conversion of point-based surfaces to polygonal surfaces with textures [Wicke et al. 2005]. Unfortunately, their global approach requires a heavy preprocess (more than 20 minutes for the Stanford Dragon) and for any even local modification, the entire preprocess has to be started from scratch.

We show in this article how to efficiently convert a point cloud in hardware-friendly polygonal structures than can be quickly and locally updated. The goal of our work is to efficiently merge 3D models represented as point clouds in state-of-the-art high quality polygonal 3D renderers, providing an additionnal layer between point-based modeling and polygonal rendering.

3 Surfel Stripping

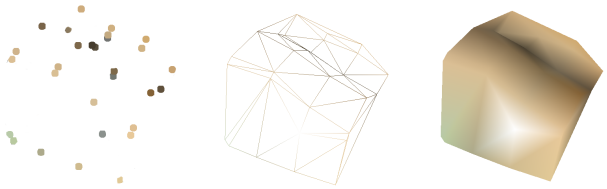


(a) (b) (c) (d) (e)

Figure 2: The different steps involved in Surfel Stripping: (a) initial surfel set, (b) corresponding Stripping Tree space-partitioning data structure, (c) a Surfel Strip is generated at each leaf of the Stripping Tree, on an *inflated* local surfel set, (d) after *decimation*, most of the overlappings have been discarded, (e) real-time rendering using 3 colored light sources.

3.1 The Surfel Strip

As said above, the basic principle of Surfel Stripping is to convert the initial PBS into a set of rendering primitives, called *Surfel Strips*, that we want to be well adapted to current 3D graphics hardware. Since *triangle strips* are one of the most efficient 3D primitives in current hardware, we define a Surfel Strip as a small 2-manifold strip of triangles that locally interpolates a subset of a PBS (see Figure 3). When the original PBS includes additional information at each point, such as normal vectors, color indices or texture coordinates, the Surfel Strip automatically inherits them on a per-vertex basis.



(a) (b) (c)

Figure 3: The Surfel Strip principle: (a) small subset of the initial surfel set, (b) local connectivity information is computed, (c) resulting Surfel Strips rendered with Gouraud shading by using a per-vertex normal and color.

This latter behavior is an important characteristic of Surfel Stripping: *all* the data that exists in the original PBS is exactly transmitted to the Surfel Strip structure. In other words, there is no compression or low-pass filtering as in usual splatting techniques [Zwicker et al. 2001; Botsch et al. 2004]. Of course, filtering *is* sometimes interesting, mainly when there is some noise in the initial PBS. But in our opinion, it is preferable to remove noise at the point-based level, with for instance [Pauly and Gross 2001], rather than spending computational effort at *each* rendering frame to low-pass filter the point set.

In addition to its ability to efficient hardware rasterization, such a localized primitive also provides a coarser granularity for many aspects of the rendering process: a large amount of operations (e.g. discarding tests for culling, see Section 3.4), can be performed at the Surfel Strip level, instead of at the point level, reducing the number of different tests to perform in a space-coherent fashion.

Once the idea of using local triangle strips for a hardware-friendly visualization of surfels is set, there are still three fundamental problems to solve to get an efficient and accurate system:

- How to efficiently generate each individual Surfel Strip? This can further be divided into two sub-problems: the efficient computation of the local connectivity and the efficient generation of the triangle strip from the connectivity.
- How to guarantee that no holes will be visible between neighboring Surfel Strips? In other words, we want an object-space hole-filling algorithm, similar to the image-space hole-filling provided by conventional splatting techniques.
- How to take benefit of the data structures constructed at loading time in order to propose an efficient rendering and in order to locally update the "visualization layer" provided by the *surfel strips*.

The next section details the algorithm that we propose to solve these two problems.

3.2 The Surfel Stripper

The *Surfel Stripper* is the core of our system: it can be seen as a blackbox that inputs a small subset S of the initial PBS and outputs a Surfel Strip (see Figure 4). A common tool used to create triangles from an unstructured set of points in an n-dimensional space is the Delaunay triangulation. Using a true 3D implementation of Delaunay to reconstruct a 2-manifold in 3D is usually not very efficient, as this process generates a lot of interior (i.e. volume) triangles that have to be removed to keep only the triangles that lie on the surface. In our case, we know that S has a small geometric extent as it represents a local subset of the PBS. So it is relatively natural to impose another constraint to S that will greatly speed-up the triangulation: S should be consistent with a *height map* representation (i.e. each point can be expressed as an elevation along the normal of an average plane). Based on this constraint, the Surfel Stripper reduces the problem to a simple 2D Delaunay triangulation, performed on S once it has been projected on the average plane Π similar to [Boubekeur et al. 2005]. This reduces the generation time of Surfel Strips by about one order of magnitude.

We define Π by the centroid of S and a normal vector that can either be obtained by using *Principle Component Analysis* on the covariance matrix of the surfel positions of S (the eigenvector associated with the minimum eigenvalue), or by simply averaging the normals of S when they are available. We use an adapted version of the *incremental randomized* Delaunay triangulation [Devillers 1998] on the projection of S, an algorithm with $\Theta(n \log n)$ complexity where n is the number of surfels in S. A typical size of n in our implementation is between 20 and 40, which offers the best overall performance for the entire Surfel Stripping process. The connectivity information generated by this 2D triangulation is then trivially applied to the original surfel set S by using the same point indices.

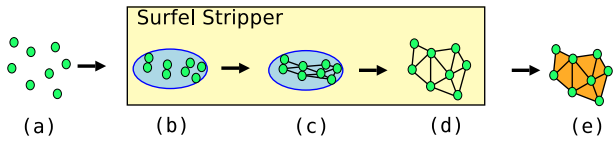


Figure 4: The Surfel Stripper process: (a) a small subset of the initial surfel set, (b) the projection on the local average plane (shown in blue), (c) the 2D Delaunay triangulation, (d) the resulting connectivity information, (e) the final Surfel Strip.

Inflate-and-Decimate: The set of surfels submitted to the Surfel Stripper is determined by the initial partitioning that will be detailed in the next section. In order to avoid holes between Surfel Strips, we improve the local triangulation of [Boubekeur et al. 2005] by proposing an efficient two pass technique that we call *inflate-and-decimate* that efficiently reduces the set of useless triangles while still maintaining a hole-free visualization.

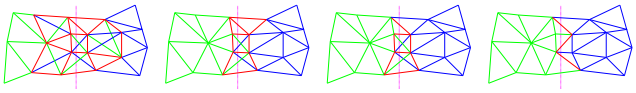
During the first pass (inflation) that is done before the Delaunay triangulation, we extend the set of surfels in a given space-partitioning cell, by including the nearest surfels from neighboring cells (see Section 3.3). The inflation factor can be conservative by including all the surfel of the neighboring space partitions, or can be optimized when a density estimation is provided. This inflated surfel set is then triangulated using a 2D Delaunay algorithm as detailed above. During the second pass (decimation), we compare the resulting triangle set with the neighboring Surfel Strips that have been generated so far and discard useless triangles in overlapping zones. This decimation pass is based on a classification of the triangles. In this classification, chosen for its low computational cost, a triangle can have one of the four following states:

- **outer**: the triangle does not share at least one surfel with the original surfel set of its associated leaf,
- **redundant**: more than one instance of the triangle is present in the overlapping zone (i.e.perfect overlapping, very frequent thanks to the Delaunay triangulation),
- **dual pairs**: the triangle forms, with a triangle sharing a common edge, the dual configuration of two triangles present in a

neighborhoring leaf,

- **valid**: in all other cases.

Discarding *outer* triangles ensures that the overlapping zone will be only a thin band of triangles in the worst case. An instance of a triangle is removed of the current inspected cell when it is *redundant*. The *dual pairs* of triangles representing geometrically the same quad have not to be kept to ensure a hole-free vizualization. The *valid* triangles are maintained and are used for the rest of the algorithm. This set of *valid* triangles, quickly detected by the use of this classification, does not certify a watertight triangulation, but reduces the number of overlappings between the small neighboring triangulations considerably. We have made this choice in order to keep the preprocessing as fast as possible. A finer classification and an additionnal local remeshing rule could lead to a watertight triangulation under some sampling criteria, but this is not useful for our visualization purpose and is also time-consuming. Indeed, one nice property of this *inflate-and-decimate* process is that it leads to Surfel Strips with boundaries that match perfectly in more than 99% of the cases. This surprisingly good result can be explained by the fact that a Delaunay triangulation is locally unique. So the same set of triangles are generated in the overlapping zones of two neighboring inflated strips and the decimation process will then perfectly remove the overlapping triangles. A typical example is shown in Figure 5. Note that using "neighboring" Surfel Strips may appear



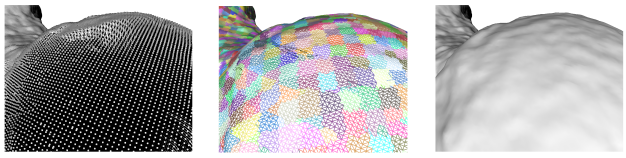
(a) (b) (c) (d)

Figure 5: The decimation pass. (a) two overlapping triangulations with a shared edge shown in red, (b) the *outer* triangles elimination, (c) the *redundant* triangles elimination, (d) the *dual pairs* elimination.

somehow in contradiction with our claim that we do not generate explicit connectivity between the strips. In fact, there is no real contradiction here because we only use the connectivity of the space-partitioning cells and do not explicitly stitch the strips together. Finally, the only annoying case where the decimation step cannot totally remove the overlapping, arises when the sampling density vs. curvature rate is too small. In this case, a different connectivity may be generated for surfels that belong to the overlapping zone of neighboring inflated strips. This is due to the very different orientation that may occur for the average planes that are computed in two neighboring cells in such high curvature areas. When this case arises, we simply keep the triangle of the inflated Surfel Strip to maintain a hole free visualization without strong artefacts (see the close-up view on Figure 12).

This inflate-and-decimate process is efficient, robust and very easy to implement. The usual approach, developed in computational geometry, to stitch boundaries of partial triangulation by computing an adjacency graph, is much more complex, requires a precise computation, and has to examine a large set of configurations to find the case where neighboring triangles must collapse. As we only seek for a hole free visualization, the proposed technique perfectly fits our requirements.

Fast stripping: In order to speed-up rendering and to limit the overhead produced by the polygonal structure, each Surfel Strip is stored as a triangle strip rather than individual triangles. Several approaches have recently been proposed to perform a direct stripping during the Delaunay triangulation. Never the less, due to the decimation step involved in our approach, it does not make sense to generate strips before the final set of triangles is actually known. We have found that the fast-stripping algorithm proposed in [Reuter et al. 2005] works extremely well to strip our small sets composed of about 50 triangles (approximatively 0.05ms to strip 50 triangles on a P4 1.8 GHz). For every leaf node of the Stripping Tree, a cache-friendly *half-edge data structure* is computed by storing the 3 half-edges at each triangle as a vector. This nicely aligns the half-edges in memory and reduces each half-edge access to one pointer de-referencing. The stripping is then done in a similar way to STRIPE [Evans et al. 1996]. Note that since the strips are computed separately in each leaf, they are constrained to the local space-partition of the leaf. Of course, this makes the strips smaller and so less optimal concerning data overhead, but as a result the strips will be more "culling-friendly" than usual long strips would have been visible from many viewpoints and thus limiting the ability of the rendering system to perform a tight hierarchical back-face and frustum culling (see Section 3.4).

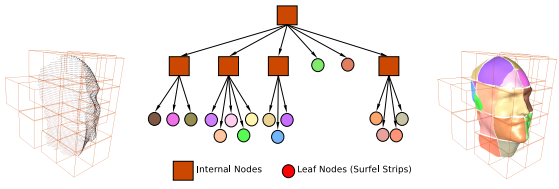


(a) (b) (c)

Figure 6: The Surfel Stripper: (a) the initial PBS, (b) the collection of Surfel Strips with random colors, almost every overlapping triangles have been discarded (c) the final Gouraud shading does not suffer from the remaining overlappings.

3.3 The Stripping Tree

After having detailed the Surfel Strip primitive and the Surfel Stripper algorithm, the last component to focus on is the *Stripping Tree* data structure that is used to efficiently subdivide the initial PBS in a way that it is consistent with the constraints required by the Surfel Stripper. Actually, almost any usual space partitioning technique (bounding sphere hierarchy, BSP-tree, kD-tree, octree) may be used, as long as a consistent split criterion can be defined. In our current implementation, we use an octree-based bounding box hierarchy. Each internal node of this hierarchy contains the bounding box of the whole set of surfels that is stored in its subtree, a cone of normal vectors used for fast culling, and 2 to 8 references to its children nodes, whereas each leaf node contains a Surfel Strip (see Figure 7). The generation of the Stripping Tree for the PBS is based



(a) (b) (c)

Figure 7: The Stripping Tree structure: (a) the partitioning of the input surfel set, (b) the adaptive tree with the Surfel Strips on its leaves. (c) a Surfel Strip is generated for each cell (with a random color for each cell).

on the main constraint of the Surfel Stripper: a Surfel Strip can only represent a height map. Consequently, we have to partition the PBS

into a collection of height maps, an approach also used by [Pauly and Gross 2001] and [Boubekeur et al. 2005]. The recursive construction is based on this local property. A node with an associated surfel set that does not satisfy this property is subdivided into 8 new nodes. The height map criterion is a simple test of the normal cone associated to each surfel set. Following [Boubekeur et al. 2005], it may also include a geometric displacement bound. This criterion can be formulated as a double condition:

$$\forall j \in [0, k_i - 1],\ n_{ij}.n_i > \delta_a,\ \delta_a \in [0,1]$$

and

$$\frac{|(p_{ij} - c_i).n_i|}{\max_k(||p_{ik} - c_i||)} < \delta_d,\ \delta_d \in [0,1]$$

with k_i the number of points of the current partition i, n_i the average normal of the surfels in this partition, p_{ij} and n_{ij} the position and the normal of the j^{th} surfel of the partition i, and c_i the barycenter of the partition. Obviously, when this criterion is not reached, the node has to be subdivided. Experimental results have shown that δ_a (angle deviation) and δ_d (geometric displacement) can be both set to 0.15 in order to avoid high distortions in the projection performed by the Surfel Stripper.

The described construction has the advantage to quickly converge towards the PBS since the local height map property is reached after less subdivision steps compared to when using BSP trees or bounding spheres hierarchies. As explained in Section 3.2, the inflate-and-decimate process used by the Surfel Stripper implies the availability of neighboring space-partitioning cells. Instead of using a topological approach based on the tree to find the neighboring cells, we have found it more efficient to simply use a geometric predicate: the epsilon box-collisions with the current cell (i.e. a test whether the box distance is smaller than epsilon) are computed between other cells in a top-down process. Then any leaf cell that passes the test is added to the list of neighbors of the current cell, and its surfels are added to the inflated surfel list. To speed-up the process, a distance threshold may be employed to add only neighboring surfels that are close enough to the current cell either using an input density estimation or a heuristic (in our implementation, the distance threshold is set to 25% of the cell diameter).

In order to guarantee a good performance of the Surfel Stripper, the space-partitioning must also ensure that each leaf of the Stripping Tree does not have to handle too many surfels. This means that in addition to the height map criterion, we also include a *population criterion* that ensures that no leaf node contains more than k surfels. We have determined experimentally that constraining $k \in [20, 40]$ provides a good trade-off for the whole preprocessing step on almost every tested model: a tradeoff between too large surfel strips (which are expensive to compute as the complexity of 2D Delaunay triangulation is not linear and does not provide good hierarchical culling), and too small surfel strips (which would lead to bad memory performance). In the case of quite uniformly sampled PBS, this population criterion also constrains the geometric extent of all resulting Surfel Strips to be very similar, as can be seen in the random color visualizations (Figures 2, 6 and 8). This feature also offers some good properties for downsampling and LODs as will be discussed in section 3.5.

3.4 Rendering Surfel Strips

The Surfel Strip collection can be directly submitted to standard graphics APIs without the use of specific vertex/fragment shaders or multipass rendering. During the rendering step, the Stripping Tree is traversed top-down, and the per-node normal cone and bounding box are used for hierarchical backface and view-frustum culling according to QSplat [Rusinkiewicz and Levoy 2000]. As illustrated in Figure 8, hierarchical backface culling can reduce the number of rendered Surfel Strips by almost 50%, even performed at the surfel strip resolution (i.e. testing the leaves, which is the extreme case for the hierarchical culling, that means testing a whole surfel strip and not the triangles individually).

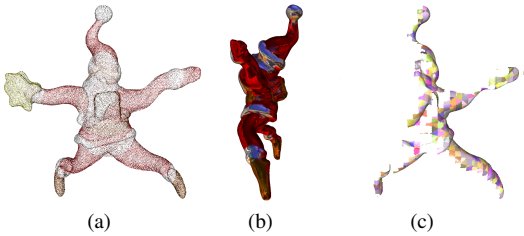


Figure 8: Hierarchical culling of the Surfel Strips: (a) the initial surfel set, (b) the Surfel Strip rendering, (c) the actual subset of Surfel Strips that has been used (i.e. non-culled) for the rendering done in (b).

3.5 Levels-Of-Detail

The main strength of Surfel Stripping is to be able to display very high resolution point clouds on very high resolution displays while providing interactive framerates, which is of major importance in many different application fields (e.g. precise archeological studies of scanned statues). On this specific point, there is currently no competitive point rendering technique that would be able to display the full resolution 3.6M antialiased textured and environment mapped point model presented in Figure 1 at 31fps on a 1600x1200 display (see discussion in Section 5). On the other hand, having only one high resolution representation of a given PBS is sometimes wasteful. Consequently, being able to switch between several levels-of-detail (LODs) would be a valuable extension of Surfel Stripping. In this section we present two different approaches for including multiresolution in the surfel stripping system.

Multi-resolution at generation step One of the main adavantages of point-based surfaces is their ability to quickly produce different levels of details of a shape. Rather than constructing a set of discrete levels of detail starting from the surfel strips at full resolution, one could prefer to take advantage of this good property of PBS by constructing a set of LOD directly on the point cloud, and then using the surfel stripping for each of these discrete levels. Near-optimal levels can be constructed using the different techniques presented in [Pauly et al. 2002]. But in order to speed up this process, we use a hierarchical simplification based on the stripping tree constructed at full resolution. This fast approach offers quite convincing results in usual cases (see Figure 9). Its only weakness is that the preprocessing time and the memory footprint is increased by about 33% as with usual mip-mapping (each inner level contains approximatively 1/4th of the strips of its child level). As usual with discrete LOD, the selection of the current level is simply based on a distance criteria.

Multi-resolution at rendering time Following [Rusinkiewicz and Levoy 2000] and [Dachsbacher et al. 2003], we have integrated a multiresolution rendering scheme in the hierarchical traversal of our structure, performing a hybrid viewpoint-dependent point-strip

(a) 40881 surfels (b) 10656 surfels (c) 2993 surfels

Figure 9: Surfel stripping for a PBS at different levels of details.

rendering, and saving time of unuseful complete drawing of too small or too far surfel strips.

Each internal node of the stripping tree carries a *representative surfel* — with position, normal and material attributes computed as an average of its children — and a bounding sphere enclosing all its leaves. During the hierarchical traversal of the stripping tree, we compute the projected size of the bounding sphere of each of the nodes. When this size is less or equal to a pixel, we draw the *representative surfel* as a single shaded point, otherwise we continue to traverse the structure top-down, performing culling as mentioned above (see Figure 10). Our experiments have shown that performing too expensive tests to decide **very** precisely when we have to render a single point (pixel) or a rasterized primitive (i.e. a triangle) cannot offer the same framerates than our approach, because of the highly optimized rendering pipeline present in today's GPU, with which it is sometimes more efficient to render a small object rather than to decide whether we have to render it. Our approach seems to be a good trade-off, since the tests performed will never reach the precision of the triangles, but will be limited, in the worst case, to test if a *whole* surfel strip leaf partition (e.g. 50 triangles) has to be fully rendered, or has to be simply replaced by a point. The *population* criterion set mentioned above (with $k \in [20,40]$) offers again a good trade-off. Contrary to the *Sequential Point Trees* [Dachsbacher et al. 2003], our approach directly handles PBS. While we do not perform the selection on the GPU, our mixed point-strip rendering reaches high framerates in practice (see Figure 1), thus letting the vertex shader instruction set free for other tasks.

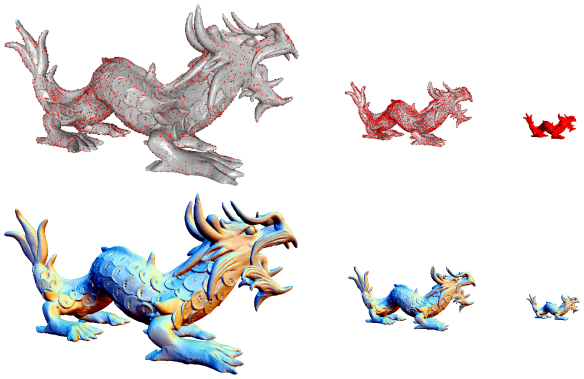
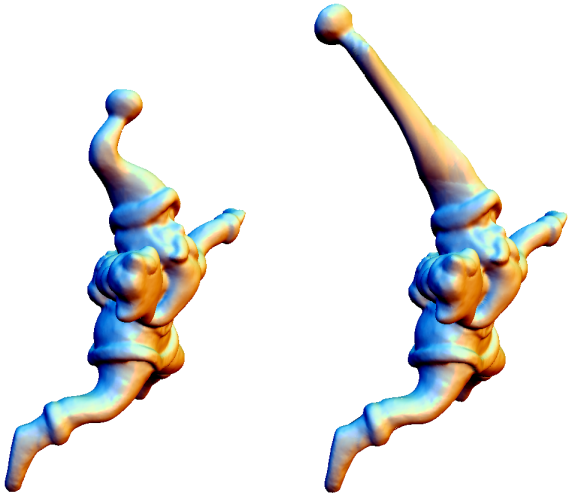


Figure 10: Top row: the surfel strip and internal nodes drawn as simple point are displayed in red. Bottow row: OpenGL rendering, the aliasing is reduced thanks to the average color and normal used for representative surfels of internal nodes.

3.6 Interactive surface deformation



(a) 75783 surfels (b) 78726 surfels

Figure 11: Interactive deformation of the underlying point point based surface.(a) The original model. (b) The deformed model locally updated.

The ability of surfel strips to be generated considering only a small local set of surfels makes possible an incremental update of the collection of surfel strips, which allows local point-based freeform deformations. For instance, let us consider the Figure 11. On the right, the Santa model (75 783 surfels) has been loaded and a stripping tree has been constructed on-the-fly to provide a direct rendering of the model.

By using conventional point-based modeling tools [Pauly et al. 2003], we have locally deformed and up-sampled the top of the model, such as shown on the right of the figure. In order to keep an interactive framerate, we keep all the surfel strips which have not been modified, and recompute the surfel strips only for the top of the model. During the interactive deformation, the modified points are classified against the stripping tree, according to the following process for each modified point:

1. Each leaf cell containing the point is marked as *modified*.
2. When the height children of a node are marked as modified, we propagate this information bottom-up in the tree, and the node is marked as *modified*.

This allows to reduce the number of full traversals of the tree: during the classification of a given point in the stripping tree, we stop the top-down traversal as soon as a marked node is encountered. After having processed all the modified points, we recompute the cells marked as modified, and update in a bottom-up fashion the *representative surfels*, normal cones and bounding spheres of internal nodes.

A slight modification of the original stripping tree generation (Section 3.3) is necessary for allowing the user to enlarge some part of the model: the original bounding box used for the octree-based decomposition of the point cloud must be over-scaled, and we ensure that all the deformations applied to the model fit inside this enlarged bounding box. Note also that during the deformation, some points can move to "empty" space. In this case, the stripping tree will be refined in location where, at the beginning, no cells were present.

The updating time is 0.18 seconds in the example of Figure 11, and the original surfel stripping performed at loading time has taken 2.67 seconds. Note that, even if it is possible, we have **not** stretched the original surfel strips of the deformed zone, but completly recomputed them. This incremental update of the stripping tree reduces the computation in the case of freeform deformations. Of course, for particularly well identified deformations, such as bone-based skinning of characters, more efficient approaches can be used to limit the number of local surfel strip regenerations. Finally, the global interactivity, during the user freeform deformation, can be increased: following [Pauly et al. 2003], a *lazy* update of our structure can be performed when deforming the object (in our case by simply "stretching" the strips for instance), and the *true* update is performed only once the deformation is finished.

4 Results

We have implemented our visualization system under Linux with OpenGL. Running times and framerates are given for an Intel P4, 3.4 GHz with an nVidia Quadro FX 4400 GPU. All tests have been done by using vertex buffers.

4.1 Visual Quality

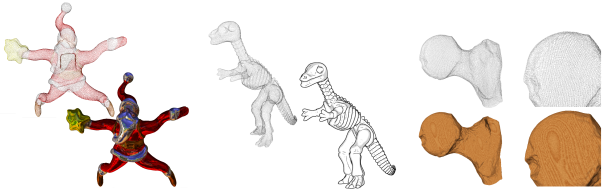
(a) (b)

Figure 12: Comparison of the visual quality: (a) the high quality EWA rendering (note the strong EWA artefacts on close-up views: lack of continuity for silhouettes and visible splat boundaries), (b) the same object at the same resolution rendered with Surfel Strips.

As pointed out by Botsch et al. [Botsch et al. 2004], Zwicker et al.'s EWA splatting [Zwicker et al. 2002] can be compared to Gouraud shading of polygons in terms of quality, since both techniques only blend colors and do not use per-pixel normal interpolation. As far as signal theory is concerned, it is true that both shading techniques have the same limit when the number of surfels/vertices grows to infinity, but actually the convergence rate is quite different: for a given number of surfels/vertices, Gouraud shading is closer to the limit shading than EWA splatting. This appears clearly on the left part of Figure 12: for the same number of points, EWA applies a stronger low-pass filtering and thus cancels much more details than the Gouraud shading provided by Surfel Stripping. Moreover, for close-up views, strong visual artefacts such as silhouette discontinuities and visible splat boundaries appear very often with EWA splatting (see the eyeball and the eyebrows on the right part of Figure 12).

Another advantage of Surfel Stripping over EWA splatting appears when rendering non-uniform point clouds: Surfel Stripping takes benefit of the Stripping Tree to perform an adaptive reconstruction in undersampled areas, and generates a hole free surface with well distributed triangles, thanks to the underlying Delaunay triangulation. On the contrary, the hole filling approach of EWA splatting is based on an adaptive per-vertex radius. So to be conservative, a large radius has to be used in undersampled areas, this produces a strong bluring effect in transition zones between undersampled and well-sampled areas.

In terms of quality for rendering of PBS, Surfel Stripping should also be compared to *Phong splatting* [Botsch et al. 2004], as both techniques propose to generate a meso-structure for the rendering of a small set of surfels. A Phong splat strongly reduces its underlying surfel set by averaging the color information and by encoding the normal variation by a quadratic function over the splat. Surfel Stripping offers much more flexibility as it interpolates (and thus preserves) *all* the position/orientation/color details included in the original point cloud, which is desirable in many applications. Furthermore, Surfel Stripping always keeps the true geometry of its surfel set, resulting in continuous silhouettes, which are not guaranteed with Phong splatting. Of course, the noise filtering property of Phong splatting may sometimes be useful, but as already said above, it can be replaced by point-based processing instead of being done at each frame.



(a) Cube mapping (b) Polygonal toon shading (c) 3D texturing

Figure 13: Surfel Strips can naturally **directly** benefit from the rich collection of polygonal rendering techniques, with many hardware-supported ones.

Modern graphics hardware offers various extensions for specific rendering tasks. As Surfel Stripping is a pure object-space approach, all these specific hardware rendering techniques are automatically available. Figure 13(a) shows the reflection produced by using cube environment mapping when rendering the strips. Figure 13(c) illustrates another important feature of Surfel Stripping: the texture resolution is totally decorrelated from the resolution of the PBS, which means for instance, that a 3D texture with a finer resolution than the point cloud can be easily rendered. This is clearly an advantage when large flat parts (that can be represented geometrically with few surfels) require a higher definition for the appearance. Such a decorrelation does not hold for splatting techniques.

Note also that the framerate does not suffer from these additional effects, since they are hardware-supported and mainly take place in the rasterization unit of the GPU. Our approach also enables a large variety of alternative polygonal rendering techniques, such as non photo-realistic ones (see Figure 13(b)).

A last advantage of Surfel Stripping compared to splatting techniques is to be perfectly adapted for an easy integration of PBS in current rendering engines. Figure 14 shows the direct use of shadow maps with antialiased **Phong Shading** in a scene that combines polygonal models and point-based models.

4.2 Performance

We achieved two different kinds of performance measurements: first, the preprocessing time required by the Stripping Tree and the generation of Surfel Strips by the Surfel Stripper, and second, the framerate that is obtained during the rendering by including the hierarchical culling and multiresolution rendering. Note that the GUI

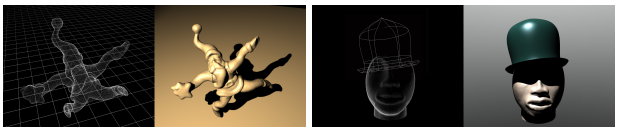


Figure 14: Surfel Stripping enables direct use of PBS in standard polygonal rendering engines. Here, two examples of antialiased Phong shading with shadow maps.

Model	Face	David	Bouddha	Asian Dragon
Points	40881	258332	543654	3609601
Surfel Strips	1612	17861	28757	89356
Preprocess	2 s	12 s	26 s	131 s
FPS	>200	167	121	31

Figure 15: Preprocessing time and rendering framerate for various models (rendering is done with antialiased 2D texture, cube mapping and 3 light sources, on a 1600x1200 screen resolution)

library that we used for the implementation has the annoying side-effect to clamp the framerate to 200 fps, so we could not precisely measure the framerate above this limit.

We performed tests on many different models up to a few million surfels (only in-core models are allowed with our current implementation) and the framerate *never* fell below 31fps on a 1600x1200 resolution, even when simultaneously activating antialiased 2D texturing, cube mapping and 3 lights sources (see Table 15 and Figure 16).

The critical step for the preprocessing is the Delaunay triangulation. Initially, we thought that the popular Fortune's algorithm[1987] would provide better results than the incremental randomized one, but for small surfel sets, better performance cannot be clearly established. The choice of an incremental triangulation also allows progressive visualization combined with progressive data transmission.

Figure 16 illustrates the robustness of Surfel Stripping for various PBS, with different densities and complex features. Our experiments have realized a perfect, crack-free and hole-free rendering for every tested model.

5 Discussion

There was a long discussion between the authors to decide whether to present the paper as a point rendering or a point-to-mesh reconstruction technique, since it is somewhere inbetween. The main argument to choose the former is that the people who use point-to-mesh reconstruction usually require watertight meshes, which obviously cannot be generated as efficiently as surfel strips. Moreover, a comparison with point-to-mesh techniques would not be fair as the state-of-the-art meshing techniques that *certify* watertight manifold (PowerCrust, SuperCocone) are much slower than Surfel Stripping. Moreover, by using an out-of-core implementation, Surfel Stripping could easily tessellate gigantic models of several hundred millions of points, which is currently *very* difficult with watertight triangulations.

To our knowledge, there are at least two previous articles that include a similar idea of local (non watertight) triangulations: the visualization system proposed for point set surfaces by Alexa et al. [Alexa et al. 2001], and *Fan Clouds* introduced by Linsen and Prautzsch's [Linsen and Prautzsch 2003]. Compared to the former, the rendering quality offered by Surfel Stripping is much higher, as it uses the position, normal and color information that exists at every single surfel, which is not the case for Alexa's technique, where the C^{-1} boundaries of the patches are apparent, since neighboring patches do not share common attributes such as normal information. Compared to the latter, both the rendering speed and the rendering quality offered by surfel stripping is higher: first, triangle strips behave better than triangle fans when hardware acceleration is considered, second, fan clouds do not offer something similar to the stripping tree to generate efficient hierarchical culling and automatic correction of undersampled areas, and third thanks to the local Delaunay triangulation (which is much more robust and much more regular than a simple k-neighborhood fan construction), the final number of overlappings with surfel strips is extremely small compared to the overlappings required to get hole filling with fan clouds.

5.1 Scalability and GPU Friendliness

The standard pipeline used in 3D graphics hardware has been developed to scale efficiently when the screen resolution is increased. Thanks to the incremental computation involved in triangle rasterization with Gouraud shading, the framerate that can be achieved by hardware rendering is only slightly affected when switching from, say, 800x600 to 1600x1200. Unfortunately, the complex per-pixel operations involved in image-space splatting techniques, such as EWA splatting, break this nice property (e.g. in the PointShop3D environment, with either software and hardware EWA renderer, the framerate is almost divided by 4 when switching from 800x600 to 1600x1200). This means that the user has to systematically find a trade-off between high-resolution rendering at low framerates and low-resolution rendering at high framerates.

This is not the case by using our approach, since it is totally based on the standard triangle rasterization, and very high framerates are obtained even for high resolutions (typically 120 fps at 1600x1200 for a PBS with 400k surfels). Another major feature of our approach, thanks to the standard pipeline, is that the rendering time of a single frame is relatively view-independent for a given number of surfels. The only component that can speed-up or slow-down the rendering time in that case, is the culling step that may discard a significantly different number of Surfel Strips from one frame to the others. This view-independent property does not hold either for image-space splatting techniques where complex per-pixel operations are involved.

But as already said above, the main advantage of Surfel Stripping compared to EWA and Phong splatting is its GPU friendliness. The process only requires one standard rendering pass, which frees graphics hardware resources to include additional visual effects by using popular multi-pass rendering tricks, such as *shadow maps*, *motion blur*, *depth of field*, etc. Actually, this was our initial goal when we developed our approach: be able to smoothly merge the rendering of PBS in current high performance 3D engines, such as the one developed for video games, with as little specific processing as possible.

5.2 Limitations

Essentially, the Surfel Stripping fails in two situations:

- **very** non-uniform sampling of the surface: in this case, the surfel stripping will not be able to fill too large holes,

- **very** dynamic surfaces, such as fluid simulations: in this case, the very frequent updates of the strips can lead to a complete regeneration of the *Surfels Strip*.

In our opinion, the first case is a reconstruction problem, and would need a geometric preprocessing, even with conventional splatting. The second limitation is still the advantage of common point-based visualization systems such as splatting, even if the efficient implementations of splatting [Botsch et al. 2005] have to deal with the update of the optimized GPU surface descriptions such as vertex buffer objets, which limits their ability to render a large number of *dynamic* points.

6 Conclusions and Future Work

In this paper, we have presented both a fast stripping method for Point-Based Surfaces and a rendering system tuned for hardware rendering at interactive framerates. Our system provides an additional object-space layer between point-based surface and polygonal rendering, represented as small triangular strips, the *Surfel Strips*, organized in an efficient hierarchical structure. The main advantage of this system is its ability to be locally generated and updated, the natural preservation of the surfel properties such as position, normal and color, and the direct reuse of conventional polygonal rendering methods.

We have shown that, in various cases, Surfel Stripping represents an efficient alternative to existing high quality rendering of PBS that have been developed in recent years, since it neither requires a specific multi-pass rendering process, nor some expensive combination of vertex/fragment shaders. Basically, our combination of hierarchical culling, multiresolution rendering and strip-based rasterization provides, in the case of high resolution, a speed-up factor of about 10 for the rendering framerate, compared to current state-of-the-art high quality point rendering techniques.

Surfel Stripping can also be seen as an alternative to complete point-to-mesh surface reconstruction offering a fast solution to import colored PBS into standard 3D applications.

The Stripping Tree has been developed to quickly space-partition a PBS and offers an efficient access to neighboring cells. At rendering time, it provides an efficient hierarchical multiresolution rendering, particularly interesting for models made of more than one million surfels. Surfel Stripping is currently not the best solution for highly dynamic surfaces, such as fluid simulation, but a convincing solution in all other cases.

We plan to continue to develop such hybrid approaches where PBS are locally "translated" in order to fully benefit from the optimized pipeline of current hardware for polygonal surfaces. As our current implementation only works for in-core models, the size of the objects that we have used for our experiments from several thousands to a few millions of surfels. So a valuable extension would be to process huge out-of-core models. Recent advances in the management of gigantic meshes [Cignoni et al. 2004] offer an interesting analysis of possible graphics pipeline optimizations. We are rather confident that some features of this work could be included in the Surfel Stripping technique to handle such huge out-of-core models.

Acknowlegment: 3D models are provided by Stanford and Cyberware. This work as been partially done at UBC (Vancouver Canada) and supported by LIGHT (INRIA associated team program).

References

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2001. Point set surfaces. *IEEE Visualization 2001*.

ALEXA, M., GROSS, M., PAULY, M., PFISTER, H., STAMMINGER, M., AND ZWICKER, M. 2004. Point-based computer graphics. *ACM SIGGRAPH 2004 Course Notes*.

BOTSCH, M., AND KOBBELT, L. 2003. High-quality point-based rendering on modern GPUs. *Pacific Graphics 2003*.

BOTSCH, M., SPERNAT, M., AND KOBBELT, L. 2004. Phong splatting. *Symposium on Point Based Graphics 2004*.

BOTSCH, M., SPERNAT, M., AND KOBBELT, L. 2005. High quality splatting on today's gpu. *Symposium on Point Based Graphics 2005*.

BOUBEKEUR, T., REUTER, P., AND SCHLICK, C. 2005. Visualization of point-based surfaces with locally reconstructed subdivision surfaces. In *Proceedings of Shape Modeling International 2005*.

CHEN, B., AND NGUYEN, M. X. 2001. POP: a Hybrid Point and Polygon Rendering System for Large Data. *IEEE Visualization 2001*.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2004. Adaptive TetraPuzzles – efficient out-of-core construction of gigantic polygonal models. *ACM Trans. Graphics (SIGGRAPH 2004)*.

COCONU, L., AND HEGE, H.-C. 2002. Hardware-accelerated point-based rendering of complex scenes. *Eurographics Workshop on Rendering 2002*.

COHEN, J. D., ALIAGA, D. G., AND ZHANG, W. 2001. Hybrid simplification: Combining multi-resolution polygon and point rendering. *IEEE Visualization 2001*.

DACHSBACHER, C., VOGELGSANG, C., AND STAMMINGER, M. 2003. Sequential point trees. *ACM Trans. Graphics (SIGGRAPH 2003)*.

DEVILLERS, O. 1998. Improved incremental randomized delaunay triangulation. In *ACM Symposium Computational Geometry 1998*.

DEY, T. K., AND HUDSON, J. 2002. PMR: Point to mesh rendering, a feature-based approach. *IEEE Visualization 2002*.

EVANS, F., SKIENA, S. S., AND VARSHNEY, A. 1996. Optimizing triangle strips for fast rendering. *IEEE Visualization 1996*.

FORTUNE, S. 1987. A sweepline algorithm for vorono diagrams. *Algorithmica 2*, 153–174.

GOBBETTI, E., AND MARTON, F. 2005. Far Voxels – a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. *ACM Trans. on Graphics 24*, 3, 878–885. Proc. SIGGRAPH 2005.

GOPI, M., KRISHNAN, S., AND SILVA, C. 2000. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Eurographics 2000*.

GROSSMAN, J. P., AND DALLY, W. J. 1998. Point sample rendering. *Eurographics Workshop on Rendering 1998*.

GUENNEBAUD, G., AND PAULIN, M. 2003. Efficient Screen Space Approach for Hardware Accelerated Surfel Rendering. *Vision, Modeling and Visualization 2003*.

HECKBERT, P. S. 1986. Survey of texture mapping. *IEEE Computer Graphics & Applications v6*, n11.

KOBBELT, L., AND BOTSCH, M. 2004. A survey of point-based techniques in computer graphics. *Computers & Graphics, v28, n6*.

LEVOY, M., AND WHITTED, T. 1985. The use of points as display primitive. *TR 82-022, Univ. of North Carolina at Chapel Hill*.

LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., SHADE, J., AND FULK, D. 2000. The digital michelangelo project : 3d scanning of large statues. In *Proc. SIGGRAPH 2000*, ACM.

LINSEN, L., AND PRAUTZSCH, H. 2003. Fan clouds - an alternative to meshes. In *Proceedings of 11th International Dagstuhl Workshop on Theoretical Foundations of Computer Vision*, Springer-Verlag.

PAULY, M., AND GROSS, M. 2001. Spectral processing of point-sampled geometry. *ACM Trans. Graphics (SIGGRAPH 2001)*.

PAULY, M., GROSS, M., AND KOBBELT, L. P. 2002. Efficient simplification of point-sampled surfaces. In *VIS '02: Proceedings of the conference on Visualization '02*, IEEE Computer Society, Washington, DC, USA, 163–170.

PAULY, M., KEISER, R., KOBBELT, L. P., AND GROSS, M. 2003. Shape modeling with point-sampled geometry. *ACM Trans. Graph. 22*, 3, 641–650.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. *ACM Trans. Graphics (SIGGRAPH 2000)*.

REN, L., PFISTER, H., AND ZWICKER, M. 2002. Object space ewa surface splatting: A hardware accelerated approach to high quality point rendering. *Eurographics 2002*.

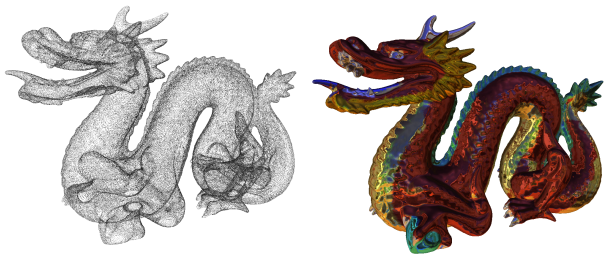
REUTER, P., BEHR, J., AND ALEXA, M. 2005. An improved adjacency data structure for fast triangle stripping. *Journal of Graphics Tools 10*, 2.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. Qsplat: a multiresolution point rendering system for large meshes. *ACM Trans. Graphics (SIGGRAPH 2000)*.

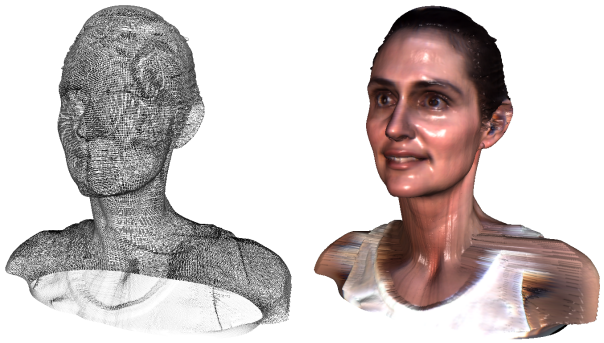
WICKE, M., OLIBET, S., AND GROSS, M. 2005. Conversion of point-sampled models to textured meshes. In *Proceedings of the Symposium on Point-Based Graphics*.

ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. 2001. Surface splatting. *ACM Trans. Graphics (SIGGRAPH 2001)*.

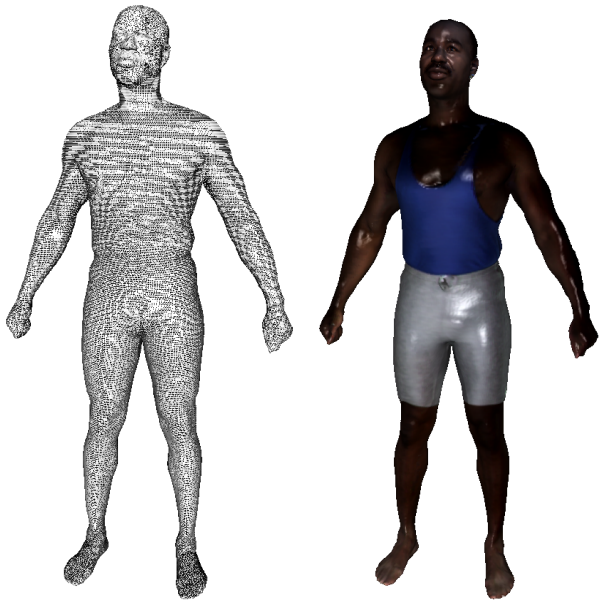
ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. 2002. EWA splatting. *IEEE Trans. Visualization, v8, n3*.



(a) Stanford Dragon with per-surfel color and cube-mapping (437 646 points).



(b) Woman face (309 737 points)



(c) Man body (146 616 points)

(d) Man face (303 382 points)

Figure 16: Realtime OpenGL rendering of surfel strips (right) converted from colored point clouds (left). *The artefacts in shoulders are not produced by the surfel stripping, they were already present in the input data.*